CS 105: Introduction to Computer Science

Prof. Thao Nguyen

Spring 2025

Materials adapted from Dave Wonnacott

Practice with Lists

(a) What are the contents of `list1` right after line 4?

(b) What are the contents of `list1` right after line 5?

(c) What are the contents of `list4` right after line 5?

(d) What are the contents of `list4` right after line 6?

```
1 list1: List[int] = [1, 2, 3]
2 list2: List[int] = [4, 5, 6]
3 list3 = list1
4 list1.insert(0,4)
5 list4 = list1.append(5)
6 list4 = list1 + list2[1:]
```

(e) What are the contents of `list3` at the end of the program?

(f) What is the boolean value of `list1 is list3` right after line 4?

(g) What is the boolean value of `list4 is list1` right after line 5?

(h) What is the boolean value of `list4 == list1` right after line 6?

(i) Describe in at least three discrete steps what the computer does “behind the scenes” on line 6.

Practice with Lists: De-/Re-composing Lists

Group work: write "find" (i.e., like the "in" operation), so that

```
find(12, [5, 12, 17, 3]) == True  and  find(13, [5, 12, 17, 3]) == False
```

You may use recursion or a loop.

Python enumerate() adds a counter to each item in a list or other iterable.

```
def find_index(target, l) -> int: # returns -1 if target is not in l
    for i, item in enumerate(l):
        # do something
```

Practice with Lists: De-/Re-composing Lists

Basic Recursive Design can be applied to lists in a number of ways

The classic (used as a fundamental programming tool as early as 1958)

- Base case: an empty list (sometimes a singleton, or one base case for each)
- Recursive case: an element (head) followed by a slightly-shorter list (rest)
- Sometimes we build a list as our result as [newElement] + simplerAnswer

Other recursive decompositions are possible:

- A slightly-smaller list followed by an element (like the classic, backwards)
- A minimum element and the other elements
- The first half and the second half
- The biggest half and the smallest half
- Those bigger than the initial element and the others

Practice with Lists, continued

Write any/all of these (note the last two may assume parameters are pre-sorted):

```
smallest([5, 12, 17, 3, 19, 12]) == 3
smaller_than(12, [5, 12, 17, 3, 19, 12]) == [5, 3]
is_sorted([5, 12, 17, 3, 19]) == False  and  is_sorted([12, 15, 175]) == True
put_in_place(55, [12, 15, 175]) == [12, 15, 55, 175]
merge_sorted([12,15,55,175], [6,8,56,1000]) == [6,8,12,15,55,56,175,1000]
```